Benjamin Hillen
CS 162
24 November 2019

Project 4: Design Reflection Document

Design Description:

The Fantasy Tournament centers around using project3_Fantasy_Combat and linked lists to construct two team Rosters and a Loser pile. The program must ask the user to play or quit, then ask for one integer that describes the size of both teams, then asks the user to choose the name and type of each fighter on both teams, then has both teams fight until one team is out of fighters.

A recover function is needed to restore some of the health a fighter has lost over the course of the tournament, which will be implemented by healing a random number bounded by the fighter's max strength and their current strength difference.

The Team class will hold the Node information for both the Rosters and the Loser containers. Each team will have a Team object that handles the Roster container maintenance, while a single, separate Team object will handle the Loser pile maintenance including adding defeated fighters to the pile.

Rosters are to be constructed in a queue-like manner, where the first fighter the user picks and names is the first fighter on the roster at the start of the tournament. After fighting, the winner of a round is cycled to the back of the roster.

The Loser pile is to be constructed in a stack-like manner, where the first fighter into the Loser pile, hereafter denoted graveyard, is the last fighter in the Loser pile.

A scoring system will be implemented by giving the winning team of a round two points, which will be kept track of through an int variable in the Menu class. The winner will be whichever team has the most points after one team has no more fighters left in their roster.

<u>Test Table:</u>

Test Case	Input	Drivers	Expected Behavior	Observed Behavior
User enters invalid main menu input	Input != “1” && Input != “2”	Menu.cpp playGame() Do...while (input != “1” && Input != “2”)	Loop asking for input until the user enters 1 or 2	Loop asking for input until the user enters 1 or 2
User tries to choose a teamSize outside FIGHTERMIN and FIGHTERMAX	teamSize > FIGHTERMAX || teamSize < FIGHTERMIN	Menu.cpp runTournament() do...while(teamSize > FIGHTERMAX || teamSize > FIGHTERMIN)	Loop asking user to enter a positive integer between FIGHTERMIN and FIGHTERMAX	Loop asking user to enter a positive integer between FIGHTERMIN and FIGHTERMAX
User tries to choose a character type not included in the presented options	Input < “1” || Input > “5”	Menu.cpp setTeam1Roster( ) setTeam2Roster( ) do...while(input > “5” || input < “1”	Loop asking user to only enter the valid integers displayed next to each character type in order to choose their character type	Loop asking user to only enter the valid integers displayed next to each character type in order to choose their character type
Fighter in either team dies		Menu.cpp fight()	Check which roster the fighter is from, then remove that fighter from the roster and add them to the front of the graveyard pile, then heal the winning fighter and send them to the back of their roster	Check which roster the fighter is from, then remove that fighter from the roster and add them to the front of the graveyard pile, then heal the winning fighter and send them to the back of their roster

One team runs out of fighters		Menu.cpp runTournament()	Announce the winning team based on team points, then clear the memory allocated during tournament runtime	Announce the winning team based on team points, then clear the memory allocated during tournament runtime

Reflection:

The hardest challenge I encountered in this project was how to handle the separation of the Roster and graveyard containers and how I would take a node from the Roster container and move it over to the graveyard container. I settled on creating one class, Team, that would hold the information on the nodes used in both containers, because each container needed to have a pointer to the next node, the previous node, and a Character* value. Then I had two head pointers, headTeam and headGrave. The headTeam pointer would be used to construct the Roster container while the headGrave pointer would construct the graveyard container. This allowed me to make three Team objects: team1LineUp, team2LineUp, and graveyard. The lineUp objects would handle Roster containers while the graveyard object would be the single Loser pile specified in the rubric.

To solve the problem of moving from one container to another, I changed the remove function of Team to return the Character pointer of the node it removed. This pointer was then passed into the addFrontGrave() function. This effectively caused the dead fighter in the Roster container to be disconnected from the Roster container and sent over to the graveyard container.

I had considered creating two separate classes, one for the Roster container and one for the graveyard pile, but found that it was simpler and faster to add the headGrave pointer to Team and treat Team as implementing multiple linked lists at once.

Class Hierarchy:

Character

↙ ↓ ↓ ↓ ↘

Barbarian	**Vampire (Overrides rollDef)**	**Blue Men (Overrides rollDef)**	**Medusa (Overrides rollAtk)**	**Harry Potter (Overrides rollDef)**